Compositing graphic with the DVG

Andrew Wojdala
Orad Hi-Tec Systems

1. Introduction

While number-crunching clusters are around for quite a while, an interest in PC-based graphic clusters started only recently and it was originally limited to multi-screen displays, with each machine of a cluster working on a separate image.

Graphics cluster architecture described in this paper allows multiple GPUs (Graphical Processing Units) to work on one image. Such scaling of the graphical performance allows more complex scenes to be displayed in high framerates.

The device that combines outputs from multiple GPUs into one image is called the compositor. There are two approaches to graphic compositing:
- build boards hosting multiple GPUs and compositor together, or
- use COTS (commercial off the shelf) graphics boards and build just the compositor.

The benefit of the first solution is that one board with multiple GPUs can be hosted in one PC, thus reducing the overall complexity of the system. However, the process of building new graphics boards with multiple GPUs is time consuming and each new GPU requires new design or at least modification of existing design of the board. Since main competitors on the market (nVidia and ATi) release new graphics chips about every 6 months, this approach does not allow fast upgrades. In addition, such boards need to have dedicated drivers and it is quite difficult to write high-performance driver for modern GPU not being its manufacturer.

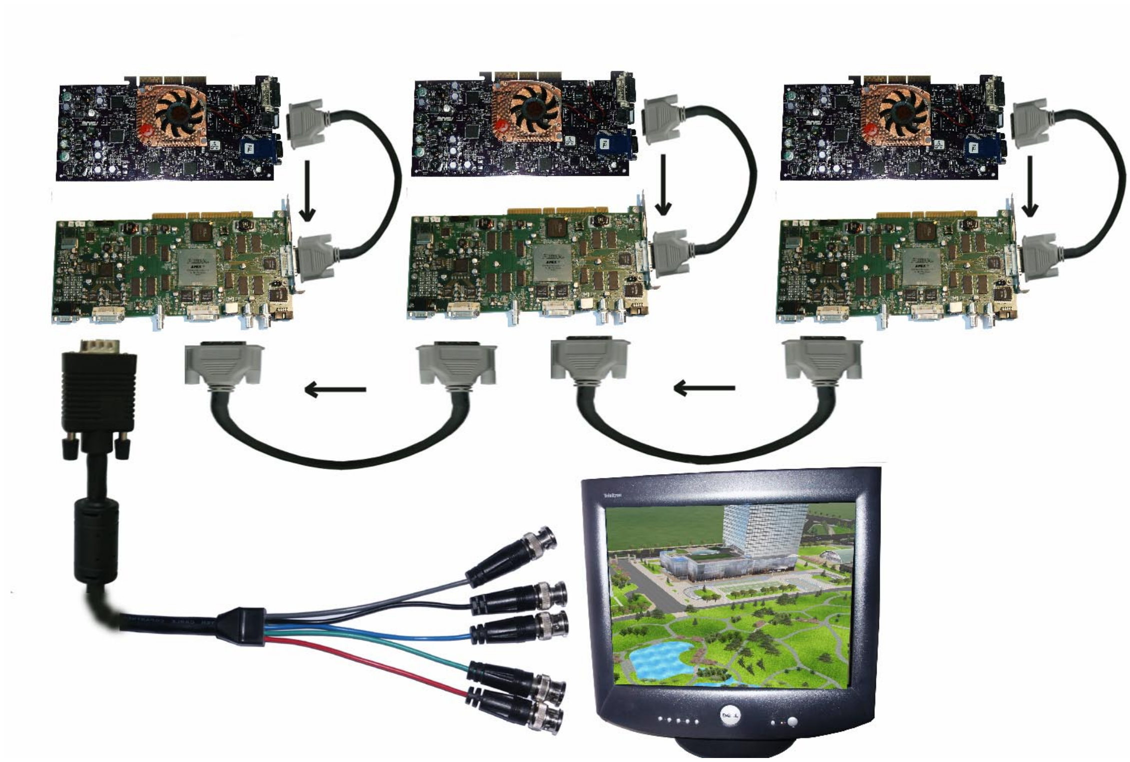
In contrast, using COTS graphics boards allows to use original drivers from the GPU manufacturer (thus enabling full capabilities of the graphics chip) and simplifies the compositor design. In particular, the same compositor – being separated from the graphics board – can be used for graphic boards of different type and manufacture. This approach was taken in graphic clusters using the DVG compositors.

Compositors separated from graphics can follow two different architectures: central and chained. Central architecture means that all graphic boards feed their outputs to central compositor unit, which produces combined output (or outputs). The drawback of this approach is that the compositor has finite number of inputs, which limits the scalability. In addition there is no flexibility regarding the number of outputs. Chained architecture means that each graphic board has its own compositor associated with it. However, the compositor is

simpler because it has only one input from the chain and one from the local graphic board. This architecture was employed in the DVG compositor.

2. The DVG Compositor

The DVG compositor is a PCI board. The picture shows how it is connected with graphics board and other compositors:



Note that the final output (VGA or DVI) is taken from the compositor and not from the graphics board. The image data is not going through the PCI but is transferred between graphic board and the compositor directly using digital “pixel bus”, thus not affecting the performance of the machine. Compositor and graphic board need to be locked together; to achieve that the graphic board is slightly modified. Compositors can be locked to incoming digital data or to analog framelock which is transferred through the chain. First compositor in the chain can work with the free-run frequency or can be locked to incoming analog or digital signal.

Outputs produced by the DVG conform to VESA standard, but it is also possible to specify all signal parameters: frequency, resolution, blanking (sync and porch) and active video. This is particularly important when using DLP projectors, which are more sensitive than CRT monitors and frequently have their own range of accepted signals. The DVG can also produce an active stereo signal (it is

equipped with an active stereo VESA miniDin-3 connector). DVG can work with graphic boards equipped with ATi and nVidia GPUs.

The heart of the compositor are FPGA chips, which process the input signals coming from preceding compositor and from the GPU and produce graphics output and signal for the next compositor. Actions taken during the processing depend on the chaining modes, which are described in the next chapter. Processing is done on the entire screen area and not on the selected section. Compositor is controlled by the driver provided with it, but the graphic driver comes from the GPU manufacturer, thus assuring the maximum performance.

3. Chaining modes

Each PC of the graphic cluster (the rendering node) has one graphic board and one compositor. For simplicity we will assume that each node runs an instance of the rendering application (more detailed discussion of issues associated with graphic cluster programming can be found in chapter 4). The DVG compositors can work in several chaining (compositing) modes, which can be mixed together. Different chaining modes address different performance bottlenecks that application might have. Five main modes are:

- Sample Division (also called "AA chaining" or antialiasing chaining"),
- Image Division,
- Time Division (TD, also called "Time Multiplexing chaining"),
- Volume Division (VD),
- Eye Division (ED or "active stereo").

In addition, Image Division compositing has 3 variations:

- Split Image Division (SID),
- Interleaved Image Division (IID),
- Added Image Division (AID).

Chaining modes describe what happens to the signals coming from preceding compositor and from local graphics as well as what signal is sent to the next compositor. Change of mode is done by the DVG driver per request of the software invoking appropriate DVG API calls. Hence, no change of connections is needed when the mode is changed.

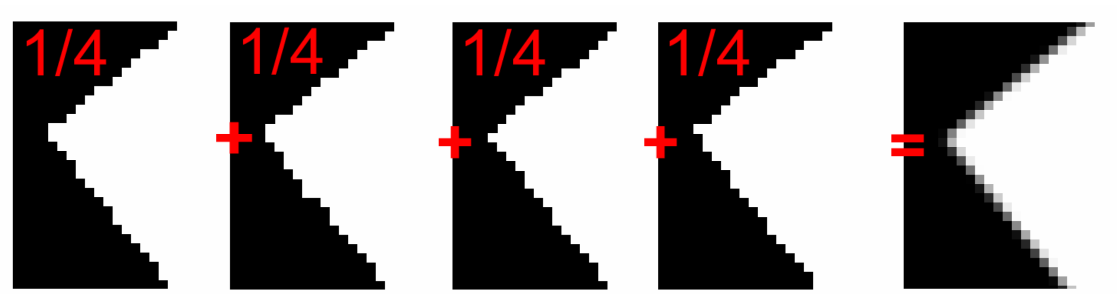
Sample Division

Currently available most powerful graphic boards based on nVidia's NV35 and ATI's R350 chips are capable of multisampling with 2, 4 or 6 samples (the latter provided only by ATi). 6 and 8 sample modes of NV35 are currently quite impractical from performance point of view. To make the things worse, 4 samples of nVidia's chips have incorrectly located samples (they are regularly spaced) and as an effect the antialiasing quality is not very good. Increasing the

antialiasing quality by multipass rendering with accumulation buffer seriously degrades the performance, so it is not a practical solution either.

Sample Division is a chaining mode designed to increase the number of samples and thus to improve the antialiasing quality. Each rendering node should draw the same view of the scene, but with different sub-pixel offsets. These offsets (describing proper location of samples) are calculated by the DVG API. Offsets are available for the application, which should draw with accordingly modified projection matrix. In each frame images generated by rendering nodes are averaged by the combiner. Two, three or four nodes can form the Sample Division chain, resulting in up to 24 samples (ATi) or 16 samples (nVidia) in the output image.

Interestingly, this technique not only increases the antialiasing quality, but has also certain performance scaling effect. Using multisampling supported by the graphics chip has negative impact on performance (the penalty varies depending on the GPU and the number of multisamples). With Sample Division it is possible for each rendering node to draw without any antialiasing (and thus with better performance) and the combiner will produce antialiased output (4 samples if the chain of 4 nodes is used):



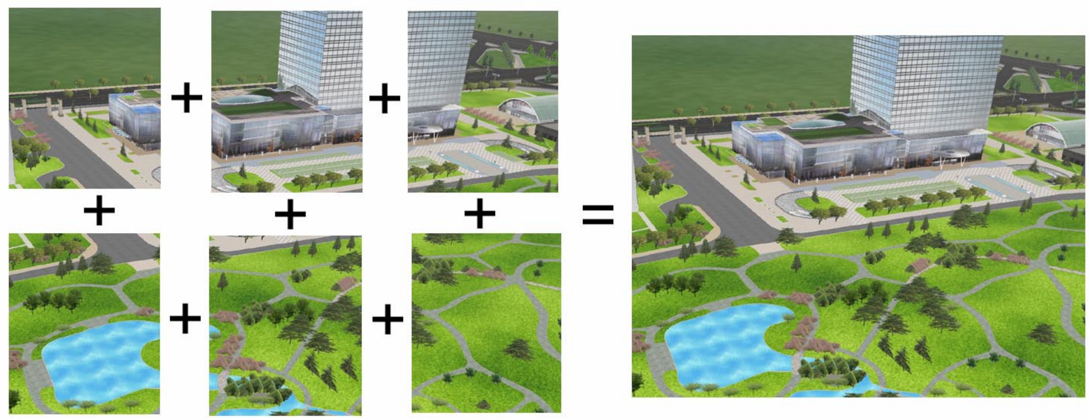
The side effect of drawing without antialiasing is reduction of the framebuffer memory, which allows more space to be allocated for textures.

Sample Division introduces one frame latency in the output image, regardless of the number of nodes.

Split Image Division

Image Division techniques are primarily targeted at high depth complexity scenes, high resolution formats, as well as scenes and rendering methods requiring 2D post-processing (such as distortion correction).

In Split Image Division the desired output screen area is divided into rectangular sections of equal size (arranged vertically, horizontally or both), which are assigned to rendering nodes:



* Scene courtesy of Chess Computers Ltd., China

On each node, DVG API calculates the projection matrix accordingly and makes it available to the application, which should use it for view culling and for drawing.

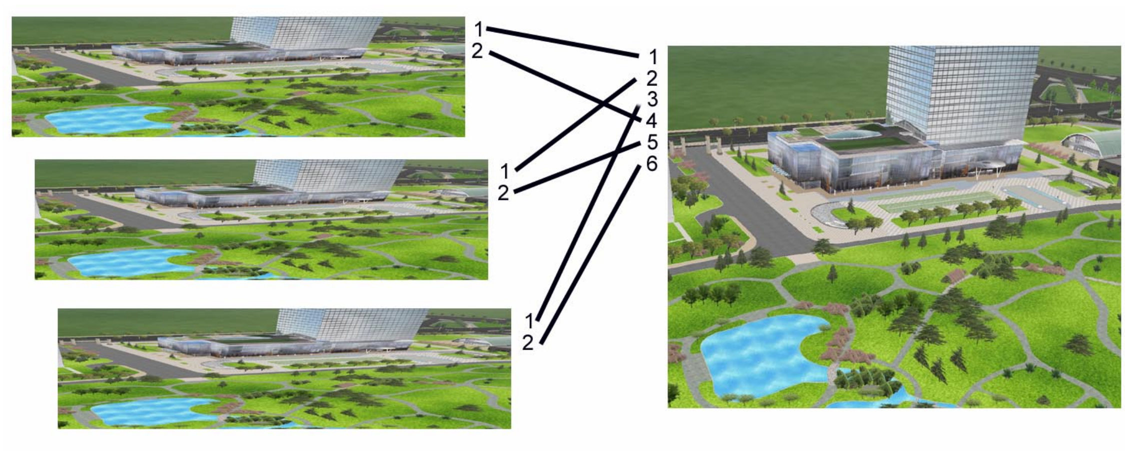
The most obvious benefit is that each rendering node draws fewer pixels, so the performance is improved whenever pixel fill rate is a limit. On the other hand, when proper view culling is done, there is also a gain in geometry rate, so overall more polygons can be drawn by the entire chain. Aside from these performance improvements, there are few not so obvious ones. Nodes are drawing in smaller resolution compared to the final output. This reduces the overhead time (the time needed for swapping front and back buffers after drawing of the frame is finished). Drawing less objects means less state changes, which is also saving the rendering time. In addition, since less graphic memory is taken for framebuffer, more is left for textures, so it is legitimate to say that using this method scales the texture space.

The performance improvement factor however cannot be precisely calculated. The behavior depends on the nature of the scene, specifically on the relation between geometrical and pixel fill complexity. If the bottleneck is pixel fill, then this type of chaining will increase performance, up to the point when the screen division is so large and the rendered areas are so small, that geometry processing complexity becomes the bottleneck and not the pixel fill. There are examples of scenes for which Split Image Division caused better than linear performance improvement because of reduced state changes and overhead time, but in practice it typically oscillates about 30-50% and varies from frame to frame. Also, different rendering nodes typically will not take the same time for rendering their sections, unlike in Sample Division chaining. As a consequence, the performance is limited by the node with the heaviest load, which requires nodes to synchronize to swap barrier (this is discussed further in chapter 4). In certain applications such as flight simulators it might make sense to divide screen only into vertical strips, because most complexity is usually concentrated on the ground, while upper part of the screen frequently shows mostly sky.

Up to 16 divisions can be defined in each direction, resulting in up to 256 sections into which the screen is divided. Graphic chip's multisampling can be used in conjunction with Split Image Division. This mode introduces one frame latency in the output image, regardless of the number of sections into which the screen is divided.

Interleaved Image Division

Interleaved Image Division can be used if uneven load of rendering nodes in SID is inconvenient for an application, under condition that the geometry complexity is not a bottleneck but depth complexity is. In this technique, each rendering node draws the entire scene in the squeezed vertically viewport. The aspect ratio is equal to the number of rendering nodes. Each node should draw image shifted by one line in respect to previous node (as in other chaining modes, DVG API calculates the projection matrix accordingly and makes it available to the application). The combiner will interleave lines from nodes to produce final output as shown on the picture below:



Formally, graphic chip's multisampling should not be used with this chaining mode, because the resulting image will not be correctly antialiased (for the multisampled image to be correct after interleaving, programming sample locations would be necessary; nVidia's GPUs do not allow it and ATi introduced it only in FireGL series boards based on R350 chip). On the other hand, when multisampling is done with 2 samples, the final image actually looks quite correct. Of course, it is possible to combine IID with Sample Division chaining to obtain proper antialiasing.

The benefit of IID is the same rendering time for all rendering nodes (because they all draw almost identical view). As an effect, this method linearly scales pixel fill rate with the number of nodes. Of course, the total gain for particular scene and view varies depending on what is the relationship between pixel fill and geometry rates. Similarly to SID, smaller viewport resolution reduces overhead time and increases the memory which can be used for textures. Also similarly,

there is one frame latency in the output image, regardless of the number of rendering nodes. The side effect of IID is that since viewport is squeezed, textures are sampled differently which might have negative visual consequences. Up to 16 nodes can be chained in this mode.

Added Image Division

In this mode each renderer draws a section of the screen, like in SID chaining. The differences are:

- the resolution of graphic window of each node must be equal to the resolution of the final output,
- the actual viewport in which an application performs the drawing has smaller resolution than the window and does not need to have the same size on all nodes,
- sizes of viewports on rendering nodes can be changed from frame to frame,
- an area outside the drawn viewport must be filled with black.

Images produced by rendering nodes are combined by simple addition, as shown below:

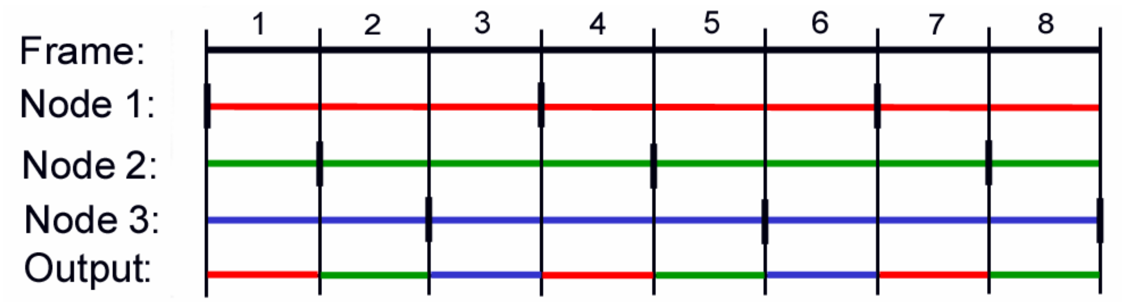
Unlike other Image Division modes, AID cannot be combined with Sample Division chaining. However, it can utilize graphic chip multisampling. Some of performance benefits offered by SID are naturally not present in Added Image Division (no reduced overhead or increased space for textures; only reduced number of state changes). But AID can be attractive for applications which are capable of dynamic load balancing by controlling sizes of drawable viewports on rendering nodes. The performance gain of this method depends on the application and potentially can be linear with the number of nodes (since view culling can be used, both pixel fill and geometry rates benefit from AID).

There is one frame latency in the output image, regardless of the number of rendering nodes. Up to 4 nodes can be chained in this mode.

Time Division

In Time Division chaining each node draws the entire scene, but the swap (not the vertical refresh) frequency of each node is *N* times lower than the output (vertical refresh) frequency (where *'N'* is the number of nodes chained in TD).

This way, nodes have *N* times more time for rendering. The role of the combiner is to multiplex between incoming signal and the local graphics. In the picture below (TD chain of 3 nodes), each combiner passes incoming signal in 2 out of 3 frames, and in third frame it outputs the contents of the local graphics (‘output’ means the final output, produced by the 3rd node):



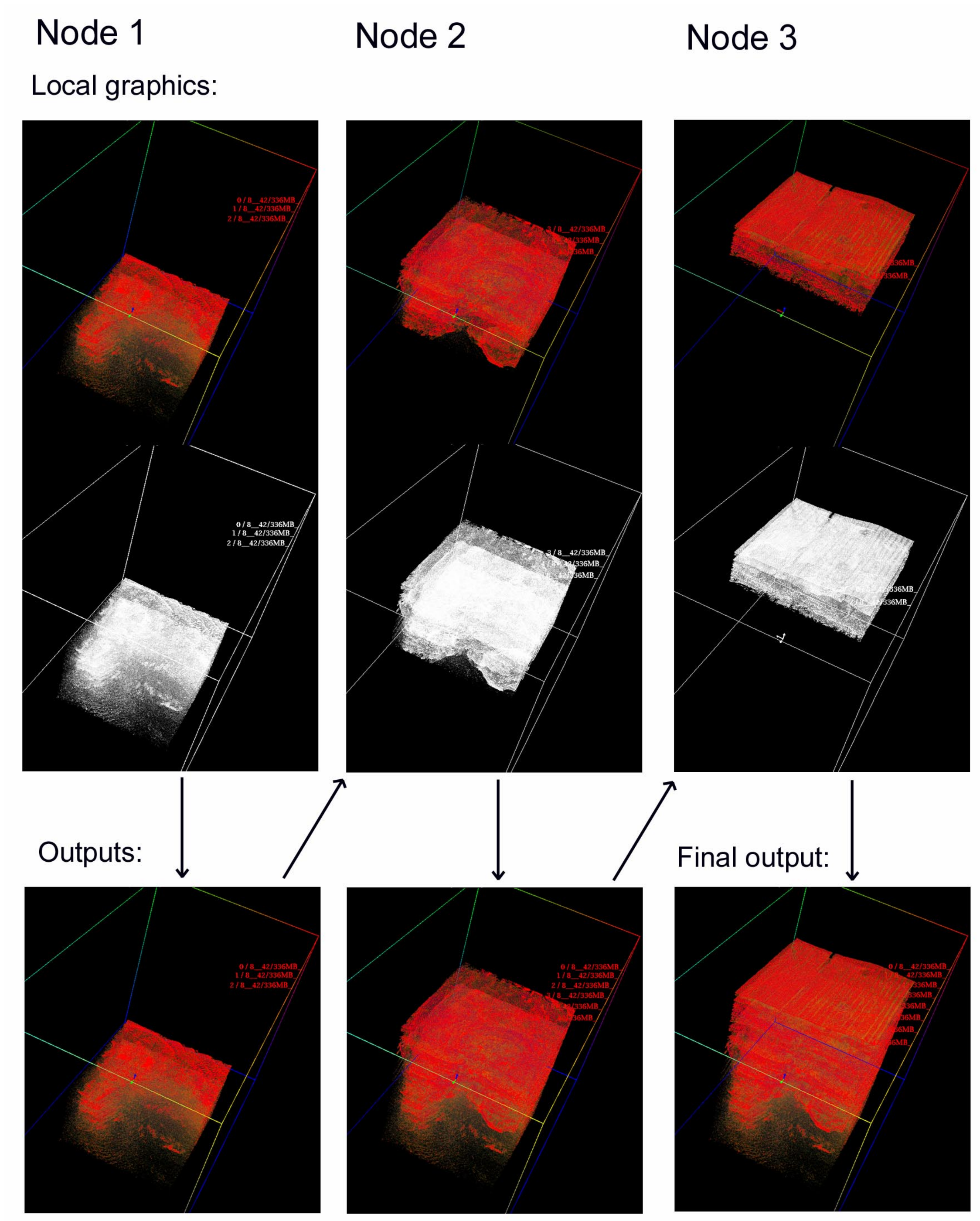
Time Division is potentially the most attractive chaining method, because the performance (both pixel fill and geometry rates) unconditionally scales up, linearly with the number of nodes. But the obvious consequence of using TD chain is the output latency increased by *N-1* frames compared to other types of chaining. In some applications (such as training devices with direct user interaction, like flight simulators) too high latency is not acceptable; in such cases it is possible to mix shorter TD chain with another chaining method (e.g., SID).

Generally it can be observed that the latency can become "harmful" in situations, where TD chaining is used for adding more complexity to the scene which initially ran in real-time, and for which the latency must remain small. It should not be however considered "harmful" if TD chaining is used for improving the frame rate of the scene, which does not run in real time. For example, if the car model runs at 5Hz then the "natural" latency (scene’s reaction to external world events) on 60Hz display is up to 12 frames (= 60 / 5). If the frequency needs to be improved to 20Hz, then TD chain of 4 nodes should be used. The natural latency will be then 3 frames (= 60 / 20). We need to add 3 frames of latency resulting from TD chaining. These frames are however 3 times longer than 60Hz frames (because they are 20Hz frames), so we need to multiply their number by 3. So, the combined latency, expressed in 60Hz frames is:
3*3 + 3 = 12
which is the same value we obtained as natural latency for original 5Hz frame rate.
Up to 256 nodes can be chained in Time Division.

Volume Division

Volume Division chaining allows to render different sections (slabs) of the scene on different rendering nodes. The condition is that slabs are depth-sorted back to front and that they do not overlap. Division of the scene and sorting is the responsibility of the application. The image generated by the local graphics is

blended with the image coming the previous renderer based on alpha generated by the local graphics. Since the combiner has access to RGB but not to alpha bitplanes, the application must assure that alpha is rendered in grayscale below the color image. Combined image is passed to the next renderer, as shown on the picture:



* Data and volume rendering software courtesy of Shell Exploration and Production

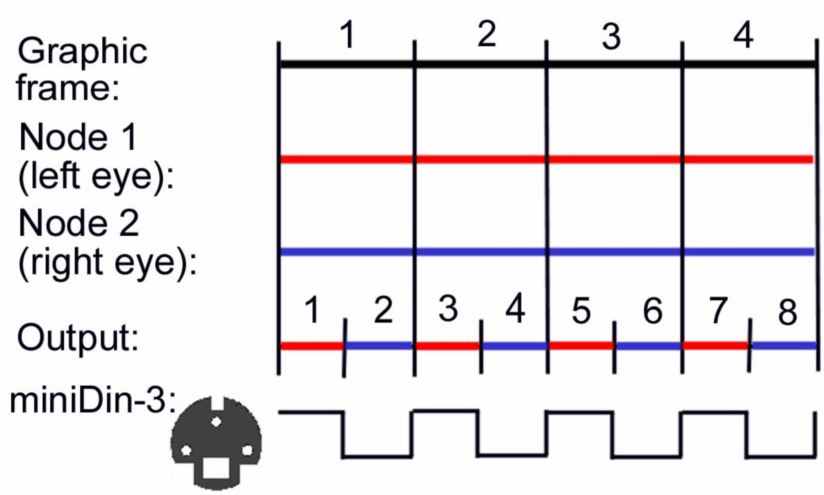
Producing alpha of course affects the performance, but actually it does not require drawing the scene for the second time. OpenGL's standard ARB extension available in both nVidia's and ATi's chips allows to copy the contents of alpha to color bitplanes in just few milliseconds (this time depends on the resolution, but is otherwise constant and independent of complexity of the scene). Another consequence of drawing alpha is that the maximum allowed vertical resolution is halved.

Volume Division is especially applicable to volume rendering applications. If datasets are properly divided between slabs, the performance gain can be linear on pixel fill rate, but the most important gain comes from data division between nodes which for large datasets can eliminate constant swapping of data having huge negative impact on performance. Certain drawback is inability to properly sort slabs in side perspective views, visible e.g. when the camera is rotating.

Volume Division cannot be used together with Sample Division or Added Image Division, but using graphic chip multisampling is possible. There is no limit for the VD chain length. The latency is one frame, regardless of the number of rendering nodes.

Eye Division

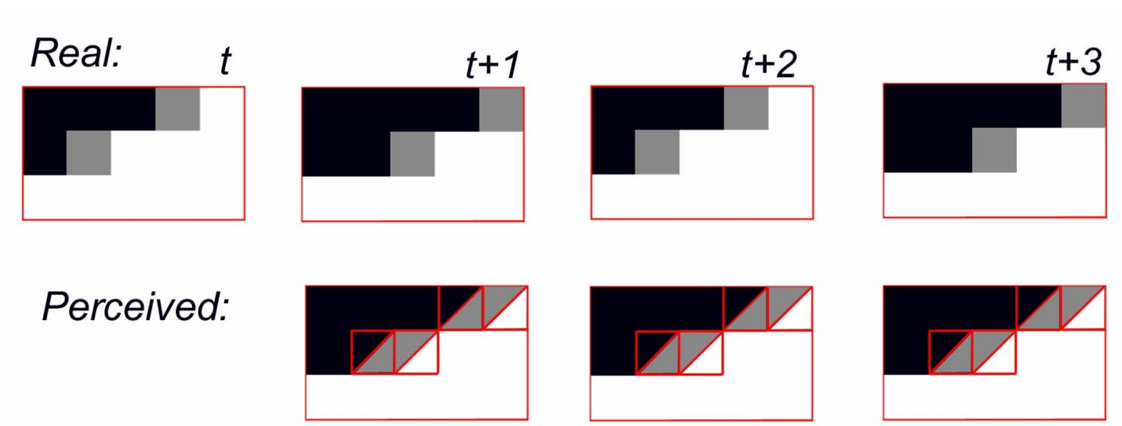
The term 'Eye Division' is used instead of 'active stereo' to describe this chaining type, as it more accurately reflects what happens in the local graphics and in the combiner. Actually, graphic does not generate the stereo signal; GPUs operate at half of the output frame rate. If the desired rate should be 120Hz, then the graphic should be drawn in 60Hz. High frequency output is generated by the last combiner in the chain together with eye selection signal for stereo emitter (or for stereo glasses), which is output by the VESA miniDin-3 connector:



The ED chain length is two by the definition, as each node should generate the graphics for one eye. It is legitimate to say that ED chain doubles the performance compared to one rendering node, because on output the scene is displayed at double frequency (if one machine was to display the same scene at double frequency, then performance would be halved). ED chain can be combined with any other chaining type, which means that multiple nodes can work for each eye.

Time-Divided Antialiasing

In addition to graphic chip multisampling and Sample Division, DVG API provides so-called Time-Divided Antialiasing (TDAA). The idea is to change sample locations in time by shifting the projection matrix by a sub-pixel. If TDAA length is two, then for each second frame the scene is drawn with different sub-pixel shift and thus antialiasing looks slightly different. There is some analogy between this technique and interlaced video (although of course full resolution image is drawn in each frame); in a way *samples are interlaced* and because of inertia of the human visual system and the display device, one can say that twice more samples are perceived than really present. As a result of using TDAA the image will jitter, which can be observed if the view is static (again, there is some analogy to interlaced video). However, because the resolution is usually higher and full frames are displayed (as opposed to fields in video), the jitter is very subtle and much less noticeable than in video. The image below shows magnified area of 5x3 pixels of a static image in TDAA length 2:

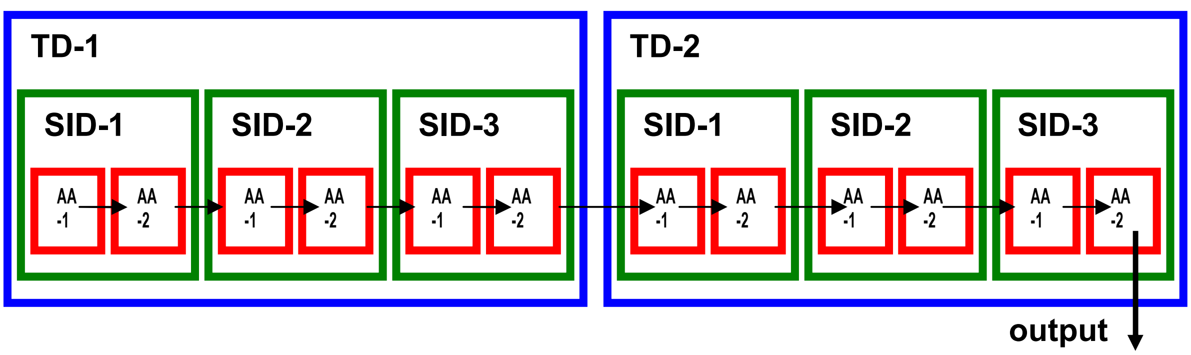


The length of TDAA is unlimited, but in practice using values bigger than two usually does not improve the perceived antialiasing quality. TDAA can be applied independently of any other chaining types, their lengths or their combinations, although using it together with Sample Division chain is not very reasonable.

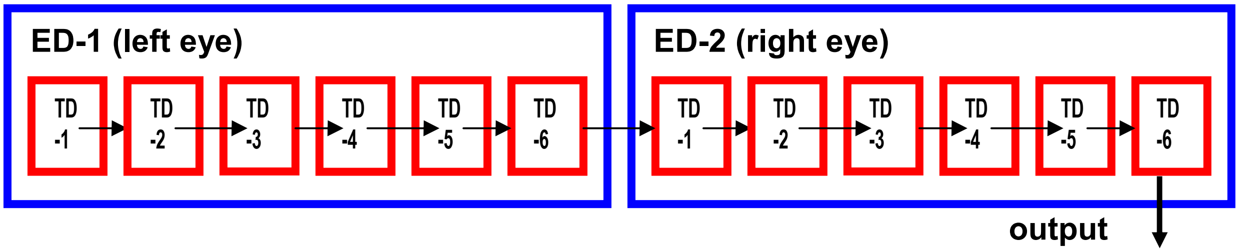
Combinations of chain types

As was already mentioned, it is possible to combine different chaining types. Configuration of the DVG chain is a software operation and does not require changes in connections. It is important to note, that the DVG chain can be configured in such a way that it produces multiple outputs (for multi-channel display or for passive stereo). It is possible because each compositor is identical and therefore has its own, fully operational output. Examples below show some of potential configurations of the chain built of 12 rendering nodes.

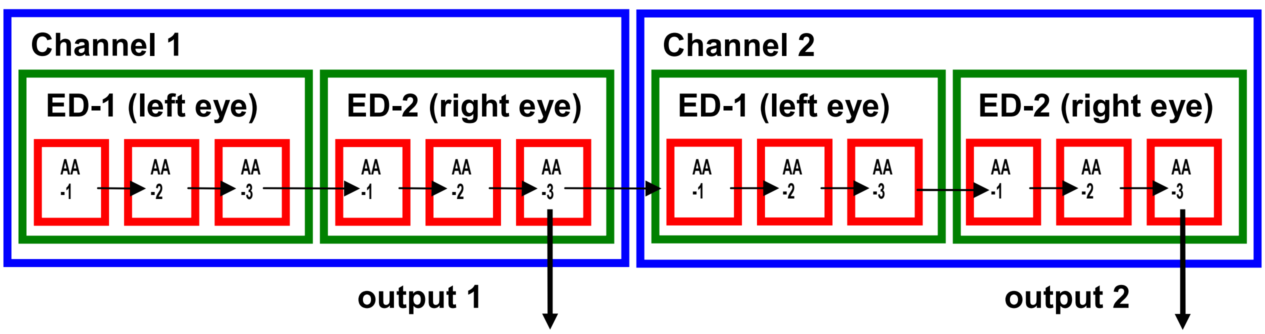
1) One mono channel; increased performance and quality; relatively low latency:
   Pairs of DVGs work in Sample Division chain (AA chain length 2), three pairs form Split Image Division chain (SID chain length 3) and two such AA-SID combinations are connected in Time Division chain (TD chain length 2). The output has:
   - same frequency as individual rendering nodes,
   - 3 times bigger resolution than rendering nodes,
   - twice more samples than single node,
   - 4.5 times better performance than single node (assuming average 50% gain of SID chain, 3 nodes are 2.25 times more powerful than one node and TD chain scales performance linearly, i.e., by the factor of 2),
   - combined latency of 2 frames.



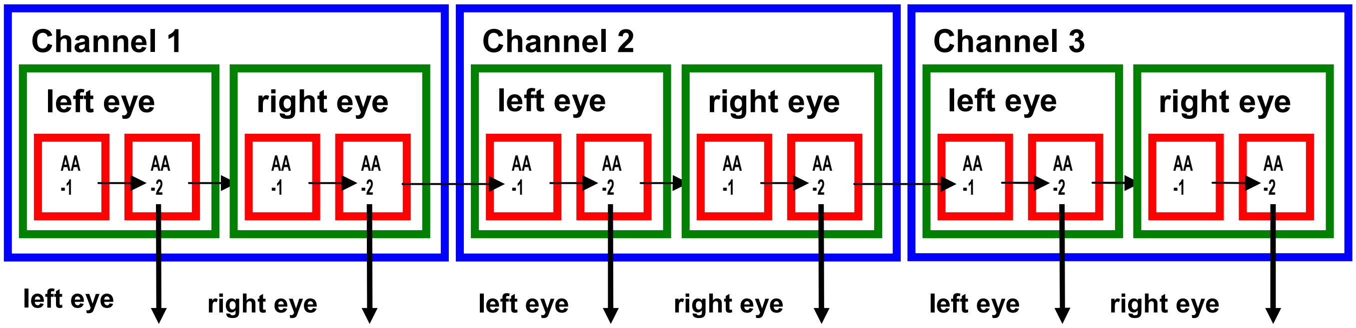
2) One active stereo channel; maximum performance; don't care about latency:
   Six DVGs work in Time Division chain (TD chain length 6), two such sextets form Eye Division chain. The output has:
   - doubled frequency of rendering nodes,
   - same resolution as rendering nodes,
   - the same samples count as single node,
   - six times better performance than single node,
   - the latency of 6 frames.



3) Two active stereo channels; maximum quality; minimal latency:
Triples of DVGs work in Sample Division chain (AA chain length 3), two triples form Eye Division chain and two such AA-ED combinations produce outputs for two independent (but synchronized) channels. Each output has:
   - doubled frequency of rendering nodes,
   - same resolution as rendering nodes,
   - three times more samples than single node,
   - doubled performance of single node (because the frequency is doubled),
   - one frame latency.



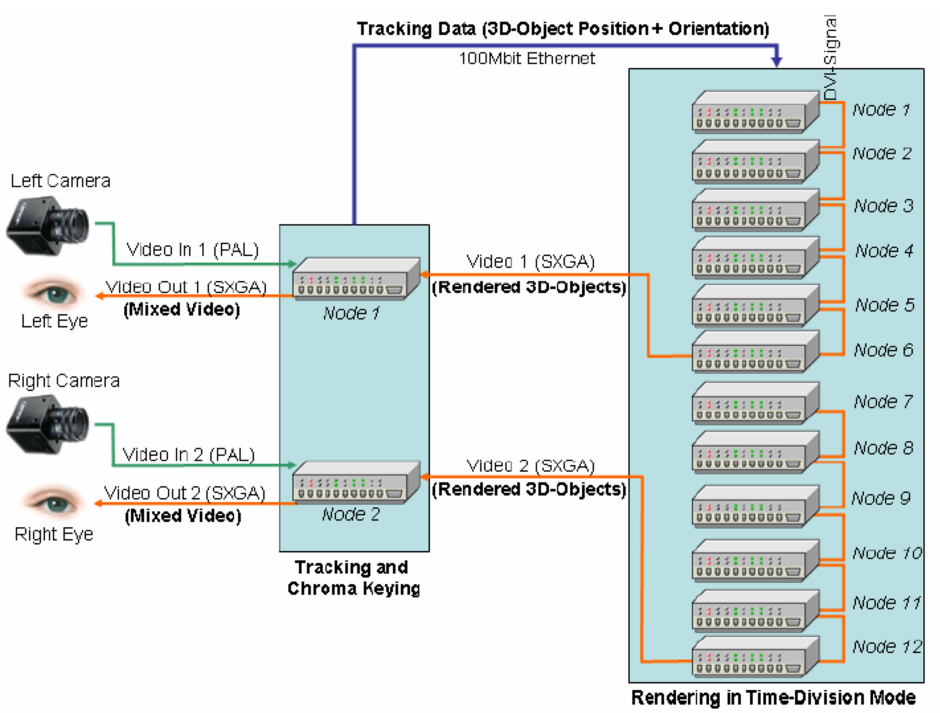
4) Three passive stereo channels; maximum quality:
Pairs of DVGs work in Sample Division chain (AA chain length 2), two such pairs produce two outputs (for left and right eye) and such combination is repeated 3 times to produce three independent (but synchronized) channels. Each output has:
   - Same frequency as rendering nodes,
   - same resolution as rendering nodes,
   - two times more samples than single node,
   - same performance as single node,
   - one frame latency.



Special chaining options

In certain applications the last rendering node can perform special function. An example is Augmented Reality (AR), where it is necessary to mix graphics with live video.

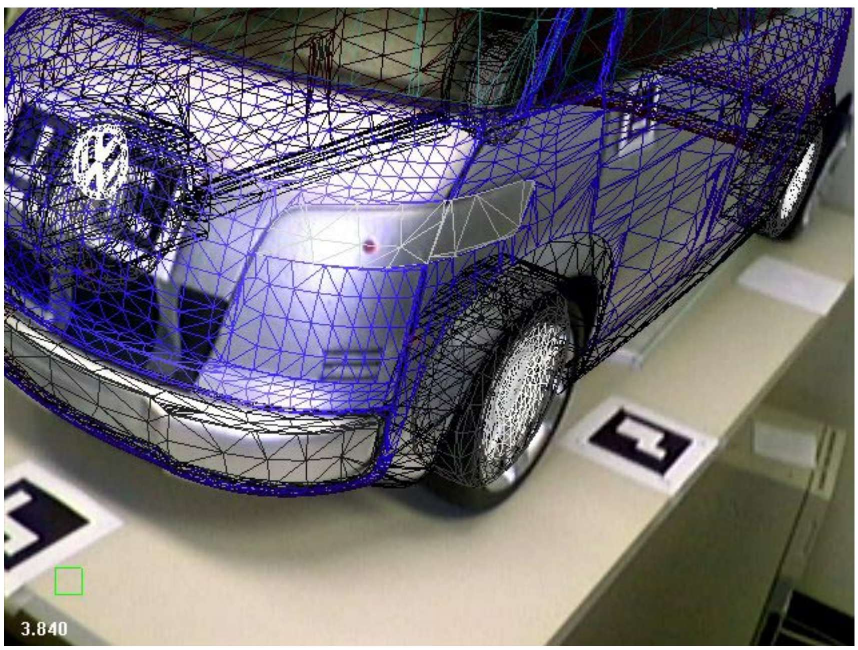
Growing number of AR applications demands high complexity of graphics elements. An example is automotive industry, where there is a need to use models of several million polygons. The picture below shows the architecture of AR system in which the stereoscopic live video is shot by two cameras attached to the HMD and producing two interlaced video signals. These signals are captured by two dedicated nodes equipped with frame grabbers. Each node is processing the captured video in real-time to retrieve the tracking data based on observed pattern and sends them by Ethernet to the rendering nodes:



* Image courtesy of Heinz-Nixdorf Institute, University of Paderborn, Germany

Capturing nodes are equipped with DVG combiners. The last node of each rendering chain is feeding its output to the combiner of one capturing node, which combines incoming graphics with video using a technique similar to chroma keying. The graphics is drawn on selected background color (e.g., blue) and the combiner of the capturing node is configured to automatically generate alpha based on this keying (background) color. This alpha is then used for blending incoming signal with full-screen polygon mapped with captured video. Blended image is output as VGA signal and is presented to the user using HMD, shutter glasses, monitor or power wall.

The figure below shows the wireframe model of a car overlaid on the video shooting the real car and markers used for camera tracking:



* Image courtesy of Heinz-Nixdorf Institute, University of Paderborn, Germany

More general example of special function of the last node is to display Graphical User Interface. Descriptions of chaining modes presented in this chapter assumed that rendering nodes produce only the real-time graphics output, and not the GUI. This is quite obvious, because otherwise it would be necessary to have GUI on each rendering node and synchronization of interaction and GUI states would be extremely difficult, if possible at all. The solution is to dedicate last node in the chain to display the User Interface and to receive interaction. Added Image Division or (Volume Division) is then used to combine incoming graphics with GUI. Of course, all rendering nodes should be aware of the GUI

windows sizes and location to adjust rendering viewport accordingly, as shown on the picture below:

It should be also generally observed that even though AA, AID and VD cannot be combined, performing AID or VD only on the last node is technically possible and can be useful (like in the example with GUI: the incoming image can be generated by the AA chain, too).

4. Programming

DVG clusters can operate under Linux and Windows 2000. In Windows environment graphics can be programmed under OpenGL and Direct3D.

There are two aspects of programming for the DVG cluster: splitting the rendering application into several rendering nodes and programming the combiner itself.

Cluster awareness

By “cluster awareness” we mean that the graphic application is prepared to run in the multi-computer environment, which requires the data flow to be synchronized between computers. Making an application cluster aware is the first step to port it to the DVG cluster. Main actions which need to be taken to achieve that are:

- separate User Interface from rendering part,
- eliminate dependency on local system (such as system date),
- eliminate random elements, or make sure they have the same seed or come from one source,
- make sure that all data that influences the look (both model itself and view) comes from one source and with the same timestamp,
- if the data is paged (e.g. from the hard disk) or if the frame rate is varying (instead of being safely real-time), then either reduce the frame rate or use the mechanism making nodes wait for each other before swapping front and back buffers (swap barrier).

Depending on how easy it is to implement the above, making an application cluster aware could be trivial or more difficult. For example, if the application only displays the scene in different views (i.e. allows only camera manipulation), then it is enough to run an instance (clone) of the application on each node and send the camera data from one source to all rendering nodes in the same moment (e.g., by Ethernet multicast), which can be usually achieved with just few hours work.

It should be also observed that applications running on PCs and supporting multiple outputs are frequently already cluster aware. Flight simulators with front and side views displayed on separate screens are one good example. Other examples are design review programs displaying wide view on power walls, programs powering CAVE or generating passive stereo view from two machines. Such applications are already capable of synchronizing data to assure continuity of views across screens.

Cloning vs. rendering slaves

Running an instance (clone) of the same application with the same data on each node is the most obvious approach which usually (if not always) assures best performance, but it is not the only one. For applications with more complex data flow it might make sense to have one master computer running the application, performing calculations and distributing the data, while slave programs running on rendering nodes are responsible only for the rendering. Data distribution can be implemented on the low level; an example is Chromium, which evolved from Stanford University WireGL project and which can be called “distributed OpenGL”. However, because commands are transmitted by the network, the consequence is usually significant performance degradation, unless mostly display lists are used and not much data is transferred in every frame. Data can be also distributed on the scene graph level (an example is Renderizer from ModViz, Inc.), which is more efficient but requires an application to be built using specific scene graph software (e.g. Performer from SGI, OpenInventor from TGS, Inc.).

Regardless of the approach (cloning or rendering slaves), separate computer is needed for interaction and GUI, because nodes perform only the rendering, as already mentioned in the section *Special Chaining Options*.

DVG API and OpenGL wrapper

Programming of the combiner is performed by invoking DVG API calls. Graphic mode setup and chaining configuration calls are typically executed once when application is started, but it is possible to change the configuration of the chain while the application is running if it is prepared for that (i.e., reads signals from API or uses OpenGL wrapper on MS Windows). It is however not possible to

change configuration from frame to frame (with the exception of changing viewport sizes in Added Image Division).

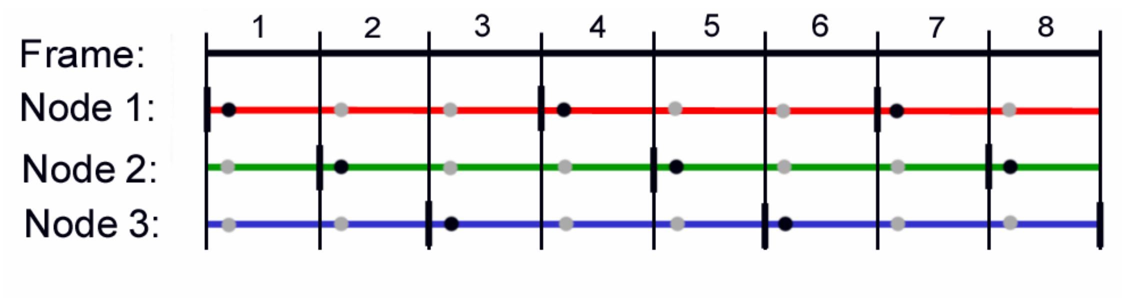
In the main loop application needs to make synchronization calls before swapping back and front buffers. DVG API allows to define so-called frame rate divider, which will hold swapping buffers for specified number of frames. For example if the divider is 2, then swap will happen twice less frequently than refresh rate. It is useful when an application is unable to keep e.g. 60Hz rate all the time and in effect the performance varies randomly between 60Hz and 30Hz. In such case it might be desired to reduce the swap rate to 30Hz using the divider 2. Swap barrier is another mechanism that allows to hold swapping until all members of the chain finished rendering of their frame (swap barrier is not supported in Time Division chain). Even though synchronization packets are exchanged by Ethernet, swap barrier is based on hardware frame counters which are always the same on all machines. Frame counters can be used by an application to synchronize data flow between rendering nodes.

Another API call that the application should make in the main loop is getting the modification for the projection matrix, so that Image Division and Sample Division chaining modes behave properly. It is possible also to get the matrix for Eye Division, though of course applications already supporting stereo perform their own calculations (similarly, application supporting division of screen into segments does not need wrapper to calculate the projection).

Equipping application with DVG API calls is not a complicated task, nevertheless is requires some changes in the code. If the code is based on OpenGL, the need for changes can be frequently eliminated by the use of the *OpenGL wrapper*. Wrapper is a DLL which intercepts certain OpenGL function calls and performs necessary actions before executing the actual call. In particular, wrapper can configure the chain during window creation without the application being aware of it. Quite obviously, glSwapBuffers() is among intercepted calls – wrapper issues non-blocking DVG API “wait for synchronization” call and only after getting sync it performs the actual OpenGL function. Wrapper also intercepts projection definition functions and modifies the matrix, thus allowing Sample Division, Image Division and Eye Division to work properly without an application taking care of it explicitly. It is even possible to configure wrapper in such a way that an application unprepared for that unknowingly supports multi-screen or stereo (both passive and active); the only drawback of supporting Split Image Division or multi-screen (tiled) display this way is that view culling will not be aware of it, so the geometry rate will not benefit from it.

From all chaining methods usually the biggest trouble is with the Time Division. The reason is quite obvious: rendering nodes work with lower swap frequency than the final output refresh rate, but data must be delivered with final output frequency and it should be multiplexed between nodes. This usually makes data synchronization more complex, although it happens that this issue can be

resolved quite easily. For example, if the application only animates the view, then it is enough to multicast camera data with output frequency to all nodes and to make sure that each node uses the most actual data:



In the picture above each node has 3 frames for the rendering. The camera data is delivered in the beginning of every frame. Only the node that starts its 3-frame long rendering in this frame uses the data (black dots). Camera data delivered while the node is in the middle of rendering (gray dots) are lost or discarded in the beginning of the next 3-frame rendering period.

We can summarize explanations presented in three sections of this chapter with three observations regarding the “application transparency”, i.e. possibility for an application to run unmodified on the DVG cluster:

- Cluster-unaware applications could run unmodified practically only when using solutions like Chromium and “rendering slaves”, which will typically impose significant performance penalty;
- Cluster-aware applications built in OpenGL will frequently run unmodified by using OpenGL wrapper;
- Using Time Division practically always requires modification of an application.